VHDL2TA: Outil de traduction en automates temporisés des circuits décrits en VHDL

Manuel d'utilisation

Equipe Architecture et Logiciels des Systèmes sur Puce (ALSOC)
Laboratoire d'Informatique de PARIS 6 (LIP6)
Unité Mixte de Recherche - UMR 7606 (CNRS - UPMC)

Abdelrezzak BARA,
Emmanuelle ENCRENAZ

12 Janvier 2011

Résumé : Ce document est un guide d'utilisation de l'outil VHDL2TA (Translation of VHDL Programs to Timed Automata) qui effectue la traduction en réseaux d'automates temporisés de circuits numériques, dont la partie fonctionnelle est décrite en VHDL et dont la partie temporelle des leurs portes est donnée par un ensemble de délais. Les réseaux d'automates temporisés générés par l'outil sont décrits dans le langage de description de l'un des outils d'analyse suivants: UPPAAL, HyTech, IMITATOR ou IMITATOR-2. Cette traduction permet de faciliter la tâche d'analyse de circuits en utilisant les techniques d'accessibilité et de model-checking. L'outil VHDL2TA a été utilisé dans l'approche d'analyse de circuits proposée dans [BBCELR.10].

Mots clés : Automates temporisés, logique temporelle TCTL, Model-checking temporisés, Accessibilité, HyTech, UppaaL, IMITATOR, Circuits asynchrones, Circuits Mémoires, Langage VHDL, Abstraction fonctionnelle (MYGALE), Abstraction temporelle (TIMEX), Architecture de la mémoire SPSMALL.

Table des matières

1. Introduction ____ 3
2. Traduction automatique en automates temporisés des circuits décrits en VHDL avec des délais 4
2.1. Le modèle de base d'automates temporisés ____ 4
2.2. Le modèle fonctionnel décrit en VHDL et le modèle temporel : ____ 4
2.3. L'algorithme de traduction ____ 5
3. Implémentation (VHDL2TA) ____ 9
3.1. Architecture et implémentation. ____ 9
3.2. Les descriptions d'entrée et sortie. ____ 10
3.3. Fonctionnalités de l'outil ____ 11
4. Mode d'emploi de VHDL2TA: ____ 11
4.1. Installation. ____ 11
4.2. Les descriptions d'entrée de VHDL2TA. ____ 11
4.3. Les commandes d'appel de VHDL2TA ____ 14
5. Exemples de tests ____ 20
Circuit Valmem latch: ____ 20
6. Remarques finales ____ 24
Références ____ 25
A. Exemples de circuits ____ 27
B. Les grammaires des descriptions d'entrée de VHDL2TA ____ 29

1. Introduction

Ces travaux concernent la vérification de circuits numériques temporisés. Ces circuits sont composés de portes logiques auxquelles des délais de propagation de fronts sont associés. L'analyse de tels circuits permet de prouver des propriétés temporelles complexes, telles que : identifier les chemins critiques, ajuster la période d'horloge du circuit analysé ou déterminer la période de stabilité des signaux d'entrées/sorties. Ces circuits sont représentés à la fois par un modèle fonctionnel décrit en VHDL et par un modèle temporel qui associe à chaque bloc fonctionnel des délais de propagation. Afin de l'analyser, ce modèle est traduit dans le formalisme d'automates temporisés dans lequel les techniques de simulation classique et de vérification par model-checking peuvent être appliquées. Avec cette démarche, on peut prouver formellement les propriétés temporelles complexes citées ci-dessus, obtenir des informations sur la marge d'erreur des caractéristiques temporelles ou bien extraire des contraintes linéaires temporelles paramétrées. Cette méthodologie d'analyse temporelle de circuits en appliquant le formalisme d'automates temporisés a été introduite par *Maler* et *Pnueli* dans [MP.95].

Pour faciliter une telle analyse, il serait pratique d'avoir un outil pour traduire automatiquement une description fonctionnelle VHDL combinée à des délais de propagation en formalisme d'automates temporisés. Ceci est l'objet de l'outil VHDL2TA. Les modèles d'automates temporisés générés par l'outil sont spécifiés dans les langages de description des outils d'analyse tels que, UPPAAL [LPY.97], HyTech [HHW.95], IMITATOR [A.09] et IMITATOR-2 [A.09]. L'outil VHDL2TA est basé sur la méthode de traduction dont les principes sont décrits dans [BBCELR.10]. Le modèle générique des portes de circuits proposé ([MP.95], [BJMY.02]) a été étendu et sera présentée dans la section suivante de ce manuel.

L'outil VHDL2TA a été développé au LIP6, Université de Pierre et Marie Curie (Paris 6), dans le cadre du projet ANR VALMEM. Son code binaire ainsi que quelques exemples de tests de circuits sont téléchargeables à partir de la page web référencée [BE.11a].

Organisation du manuel : la deuxième section de ce manuel présente le modèle temporel adopté pour représenter à la fois la partie fonctionnelle et la partie temporelle des circuits, ainsi que les principes de traduction que l'outil VHDL2TA implémente. La section 3 décrit l'architecture de l'outil et ses caractéristiques. Dans la section 4, on présente le mode d'installation et d'emploi de l'outil. La section 5 est consacrée à la présentation de quelques exemples de tests de circuits en utilisant : l'outil VHDL2TA (pour générer le modèle temporel), les outils de model-checking UPPAAL ou HyTech (pour vérifier le modèle temporel) et l'outil de synthèse de contraintes temporelles IMITATOR-2 (pour synthétiser des contraintes sur les paramètres de délais qui assurent toujours le bon fonctionnement du système). La section 6 conclue ce document. Les annexes ajoutent des précisions techniques. L'annexe A décrit le code source des descriptions de sortie de quelques exemples de circuits vus dans la section5, l'annexe B les grammaires des descriptions d'entrée de l'outil VHDL2TA.

2. Traduction automatique en automates temporisés des circuits décrits en VHDL avec des délais

Après une brève présentation du formalisme d'automates temporisés, on décrira le fragment VHDL et le modèle de délais considérés pour représenter, respectivement, la partie fonctionnelle et la partie temporelle du circuit analysé. On décrira, ensuite, l'algorithme de traduction en commentant les choix de modélisation.

2.1. Le modèle de base d'automates temporisés

En résumé, un automate temporisé [AD.94] est une machine à états finie enrichie avec des horloges qui évoluent toutes avec la même vitesse et qui peuvent être mises à zéro. Un état est un couple *(l,v)* où *l* est une location (ou état de contrôle) et *v* est une configuration de valeurs des horloges. Chaque location est associée à une conjonction de contraintes linéaires sur les horloges, appelée un invariant. Un état *(l,v)* possède deux types de transitions :

- une transition discrète étiquetée par un label d'action, notée *e*, vers l'état *(l',v')* si *v* satisfait la contrainte associée à *e*, appelée *garde* et *v'* est obtenue à partir de *v* en remplaçant certaines horloges par zéro (les horloges apparues dans la liste des mises à jour de la transition *e*).
- une transition d'écoulement du temps de délai *t* à l'état *(l,v') si* $v' = v+t$ *et pour tout* t' *t.q* $0 \leq t' \leq t$, $v+t'$ satisfait l'invariant associé à *l*.

La composition parallèle de deux automates temporisés est obtenue par la synchronisation de deux ou plusieurs transitions étiquetées sur la même action. Un réseau d'automates temporisés est une composition parallèle de deux ou plusieurs automates temporisés. Ce dernier représente le modèle global du système analysé.

Les réseaux d'automates temporisés sont bien analysés avec les techniques de model-checking. Divers outils d'analyse ont été développés dans cette discipline, tels que : les outils UPPAAL et KRONOS [Y.97] qui effectuent la vérification automatique des propriétés temporelles exprimées dans la logique TCTL, l'outil HyTech qui permet l'analyse d'accessibilité et l'outil IMITATOR (et IMITATOR-2) qui permet de synthétiser des contraintes linéaires sur des paramètres temporelles qui assurent le bon fonctionnement du système analysé.

2.2. Le modèle fonctionnel décrit en VHDL et le modèle temporel :

Les programmes VHDL représentent la partie fonctionnelle du circuit à analyser. On prend en considération les descriptions Data Flow constitués : d'affectations concurrentes définissant les blocs combinatoires et des processus définissant les blocs séquentiels (latchs, points mémoires ou buffers). Chaque instruction concurrente permet d'affecter une nouvelle valeur au signal de sortie du bloc correspondant. Les informations temporelles (partie temporelle du circuit à analyser) sont dissociées de la description VHDL : on associe à chaque bloc, deux intervalles de délais de propagation des fronts montants et descendants de son signal de sortie. Ces intervalles de délais sont parfois extraits des tables de timing associées aux blocs du circuit à analyser. C'est le cas de l'architecture de la mémoire SPSMALL [CEFX.06]. Cette dernière est une architecture complète abstraite par le LIP6 dans le cadre du projet ANR VALMEM (en utilisant l'outil d'abstraction

fonctionnelle MYGALE [LBG.98] et l'outil d'abstraction temporelle TIMEX [?]) en partant d'une description au niveau transistors.

2.3. L'algorithme de traduction

Les principes généraux de l'algorithme sont les suivants :

Dans ce modèle, chaque bloc de la description VHDL est traduit en automate temporisé qui modélise la propagation des fronts montants et descendants des signaux d'entrée sur la sortie de la porte en respectant les délais de propagation induits par le bloc représenté. Avec ce modèle, les fronts des signaux sont identifiés et propagés sur les sous parties du circuit. Cependant, la détection de ces fronts nécessite l'utilisation de variables afin de mémoriser la valeur du signal avant et après l'action d'écriture. Ceci simplifie la tâche d'analyse ultérieure.

Dans ce modèle, chaque automate temporisé possède une horloge locale qui permet de mesurer le délai de propagation. On distingue deux intervalles de délais pour la propagation des fronts de sortie (le délai de propagation de chaque bloc est compris dans un intervalle $[d(s_{up}),D(s_{up})]$, ou $[d(s_{dn}),D(s_{dn})]$ selon le sens du front de sortie propagé). Ces intervalles sont décrits par des paramètres. Ce modèle une extension du modèle bi-bounded inertial delay décrit dans [MP.95] et [BJMY.02]. Dans ce dernier modèle, le temps de propagation d'un front d'une entrée d'un bloc vers sa sortie est compris dans un unique intervalle $[d,D]$. En fait, on ne distingue pas le sens du front propagé.

L'algorithme proposé est décrit formellement comme suit :

- Identification et création des variables globales : pour chaque bloc (combinatoire ou séquentiel) associé à un signal s, créer :
  - une horloge locale x_s.
  - une variable booléenne v_s.
  - des labels de synchronisation s_{up} et s_{dn}.
  - des bornes de délais $l(s_{up})$, $u(s_{up})$, $l(s_{dn})$, $u(s_{dn})$.
- Pour chaque affectation concurrente associée à un signal *s*, créer *TA(s)* (voir la sous-section 2.3.1).
- Pour chaque processus associé à un signal *s*, créer *NTA(s)* (voir la sous-section 2.3.2).
- Instancier les paramètres de délais (voir la sous-section 2.3.3).
- Créer *TA* (ou *NTA*) pour l'environnement des signaux d'entrée.

La composition des automates créés constitue le modèle globale associé au circuit à analyser. Les traces temporelles du modèle simulent bien les propagations des fronts des signaux sur les blocs combinatoires et séquentiels du circuit. L'ensemble des traces temporelles est complet : chaque exécution potentielle du circuit est représentée par une trace temporelle dans le modèle. L'ensemble des traces temporelles est une sur-approximation de l'ensemble d'exécutions du circuit : ceci est dû aux approximations des délais des blocs du circuit par des intervalles. On peut donc avoir des traces temporelles qui ne correspondent à aucune trace d'exécution du circuit. Cette sur-approximation est réduite par l'utilisation de deux intervalles de délais très serrés avec lesquels on distingue les fronts montantes des fronts descendants de la sortie, plutôt que de les associer en un seul large intervalle (l'enveloppe convexe de l'union des deux intervalles).

2.3.1. Représentation des blocs combinatoires : chaque bloc combinatoire qui affecte à un signal *s* une nouvelle valeur, est représenté par un automate *TA(s)* composé de trois états : *stable(s)*, *rising(s)* et *falling(s)*. L'état *stable(s)* est un état passive dans lequel la valeur du signal *s* est conforme à la configuration d'entrée (même si cette dernière change). L'état *rising(s)* (resp. *falling(s)*) est un état de calcul sur lequel on passe à chaque fois qu'une nouvelle configuration d'entrée qui induit un nouvel front montant (resp descendant) sur la sortie *s*, après un délais $\delta(s_{up})$ compris entre les deux bornes $l(s_{up})$ et $u(s_{up})$ (resp. $\delta(s_{dn})$ compris dans $[l(s_{dn}),u(s_{dn})]$). On peut transiter entre les deux états *rising(s)* et *falling(s)* dans le cas où deux fronts successifs sur les configurations d'entrée, qui induisent deux fronts opposés sur la sortie, surviennent avant que la sortie pour le premier front se stabilise. Avec cette représentation, seuls les fronts significatifs sont calculés et passés à travers les blocs du circuit. Ceci permet d'avoir des calculs optimisés.

La figure 1 présente l'automate temporisé associé à l'affectation concurrente *s <= f(e_1, ... , e_m)* qui affecte le signal *s*. Les transitions de l'ensemble <*b*> (resp <*b'*>) réfèrent à l'occurrence d'un front sur l'un des signaux de l'ensemble *e = {e_1, . . . , e_m}*, induisant un front montant (resp descendant) sur la sortie *s* qui était stable sur la valeur 0 (resp. 1) après un délai $\delta(s_{up})$ (resp. $\delta(s_{dn})$): ici, on transite vers l'état de calcul *rising(s)* (resp. *falling(s)*), dans lequel on attend l'écoulement du délai de propagation $\delta(s_{up})$ (resp. $\delta(s_{dn})$) (représenté comme un invariant d'état : $x_s \leq u(s_{up})$ (resp. $x_s \leq u(s_{dn})$)) avant de passer vers l'état stable *stable(s)* avec un nouveau front montant (resp. descendant) sur la sortie *s*. La transition <*a*> (resp. <*a'*>) réfère à la production d'un front montant (resp descendant) sur la sortie s une fois que le délai de propagation $\delta(s_{up})$ (resp. $\delta(s_{dn})$) est écoulé (représenté comme une garde de transition: $x_s \geq l(s_{up})$ (resp. $x_s \geq l(s_{dn})$)) : ici la sortie se stabilise sur la valeur 1 (resp 0). Les transitions de l'ensemble <*c*> (resp. <*c'*>), réfèrent à l'occurrence d'un front sur l'un des signaux d'entrée, pendant que le modèle est dans l'état du calcul *rising(s)* (resp. *falling(s)*) pour la production d'un front montant (resp. descendant) sur la sortie, qui induit un changement sur cette valeur de calcul de la sortie *s*. Les autres transitions réfèrent à l'occurrence des fronts sur les signaux de l'ensemble e qui n'induisent pas du changement sur la sortie *s* (les transitions qui bouclent).

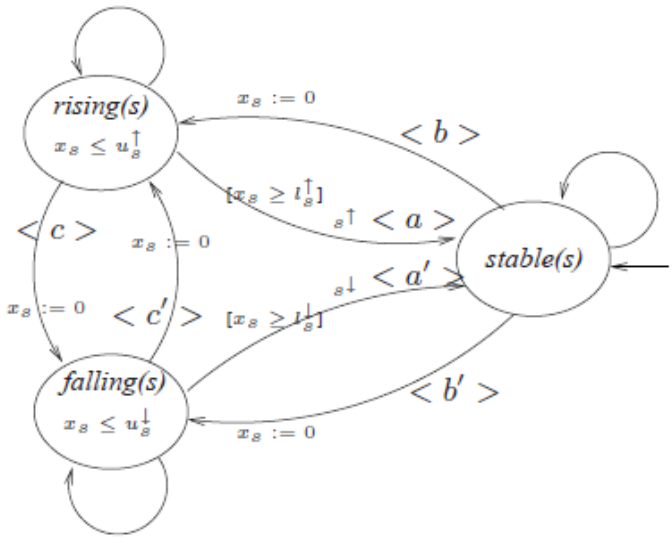


Figure 1: l'automate représentant l'affectation d'un bloc combinatoire

2.3.2 Représentation des blocs séquentiels : chaque bloc séquentiel est représenté par un processus. Le format général des processus décrits en VHDL qu'on modélise est donné comme suit.

```
Process (e_1, . . . , e_m)
   begin
         if g_1
                  then s <= f_1 () ;
         else-if g_2
                  then s <= f_2 () ;
                              .
                              .
         else-if g_n
                  then s <= f_n () ;
    end
```

Comme on peut le constater, ce processus prend en entrée un ensemble de signaux d'entrée, noté *e = {e_1, . . . , e_m}*, et attribue une nouvelle valeur au signal de sortie *s* si l'une de ses gardes est satisfaite par la configuration courante des signaux d'entrée. On peut voir ce processus comme une collection de *n* commandes avec des gardes exclusives, dont les variables sont les signaux qui appartiennent à la liste de sensibilité du processus (l'ensemble *e*) qui attribuent des nouvelles valeurs au signal de sortie *s* du processus. Une commande avec une garde *i, tel que i in {1, . . . ,n}*, est modélisée par : *[not g1 and . . . and gi] => s <= fi (e)*, où *s <= fi(e)* est l'affectation du signal *s* avec la valeur de l'évaluation de la fonction booléenne *fi(e)*. Cette affectation aura lieu si la condition *gi* est satisfaite et tous les autres conditions *gk tel que k < i*, sont fausses.

Cette représentation est modélisée par un produit de *n+1* automates connectés avec des variables auxiliaires *gi* représentant les valeurs de vérité de leur garde correspondante. Le système est composé de : *n* automate *TA(gi)* qui évalue chaque garde *gi* et un automate *TA(s)* qui affecte le signal *s*.

L'évaluation de chaque grade *gi* est représentée par un automate *TA(gi)* présenté sur la figure 2. Cet automate contient deux locations : l'état stable *stable(gi)* dont la valeur de la garde est conforme aux valeurs des signaux de l'ensemble *e* (correspond à la valeur instantanée de la garde, soit vraie, soit fausse), l'état actif *active(gi)* de délai nul vers lequel (auquel) on transite (via les transitions de l'ensemble <*g*>), de l'état stable *stable(gi)*, à chaque fois qu'il y a un front sur l'un des signaux de l'ensemble *e* qui induit un changement sur la valeur de la garde *gi*. Le changement de la variable auxiliaire v_{gi} *(* $v_{gi} = 1 - v_{gi}$*)* représentant la garde *gi*, dans le modèle, est effectué dans l'action des deux transitions uniques et urgentes (les transitions de l'ensemble <*g'*>) sortante de l'état actif *active(gi)* vers l'état stable stable*(gi)* (l'une est associé à la valeur vraie et l'autre est associée à la valeur fausse).

Évidement (comme dans l'état stable *stable(s)* de l'automate présenté dans la section précédente), tous les fronts sur les signaux de l'ensemble *e*, qui n'induisent pas de changements sur la valeur de la garde *gi*, sont modélisés par des transitions qui bouclent de l'état stable *stable(*g_i*)* vers lui-même.

L'affectation du signal s est représenté par un autre automate (présenté dans la figure 2, pour la lisibilité, tous les labels de synchronisation et les gardes de transitions ont été omis). Toutes les affectations produisant un front montant sont regroupées dans un unique état de calcul *rising(s)*, tandis que les affectations produisant un front descendant dans un état de calcul *falling(s)*. Ce regroupement réduit la taille de l'automate, mais induit la réunion de tous les intervalles de délais associés à ces affectations (l'enveloppe convexe de la réunion) : en fait, les bornes supérieures des intervalles de délais $\delta(s_{up})(i)$ (resp. $\delta(s_{dn})(i)$) sont représentées comme des invariantes d'états de la

forme $x_s \leq u(s_{up})(i)$ (resp. $x_s \leq u(s_{dn})(i)$), et le maximum de ces bornes sera représenté comme l'invariante de l'état *rising(s)* (resp. *falling(s)*) de la forme $x_s \leq max\ (u(s_{up})(1), \ldots, u(s_{up})(n))$ (resp. $x_s \leq max\ (u(s_{dn})(1), \ldots, u(s_{dn})(n))$) (le minimum des bornes inférieures est représenté comme une condition $x_s \geq min(l(s_{up})(1), \ldots, l(s_{up})(n))$ (resp. $x_s \geq min(l(s_{dn})(1), \ldots, l(s_{dn})(n))$) dans la garde de la transition vers l'état stable, qui déclenche un front montant (resp. descendant) sur la sortie *s*). D'autre part, on distingue deux états stables sur la sortie s : l'état *stable(s)* qui correspond à la stabilité de la sortie quand au moins l'une des gardes *gi* est évaluée à vrai (dans ce cas *s* est stable sur la valeur de *fi(e) t.q i* est l'indice minimum), tandis que la location *stable-close(s)* correspond au cas où aucune garde *gi* n'est satisfaite et la sortie *s* est verrouillé (le processus est fermé).

Le déclenchement des transitions est causé par le changement des variables intermédiaires associées aux gardes et aussi par le changement des signaux apparus dans les affectations de processus. Donc, les transitions sont sensibles implicitement aux changements des signaux de sensibilité du processus (l'ensemble *e*). Les transitions des ensembles <*a*>, <*a'*>, <*b*>, <*b'*>, <*c*>, <*c'*>, de la figure 2, sont similaires à celles du modèle des portes combinatoires présenté précédemment (décrit dans la figure 1), lorsque le processus correspond à un registre ouvert. Les transitions de l'ensemble <*d*>, <*d'*> et <*f*> se produisent à chaque fois qu'un changement sur la valeur de la garde *gi* de la valeur 1 à 0 est apparue, et toutes les autres gardes *gk tel que* $k \neq i$ sont à 0. La sortie reste stable sur l'ancienne valeur calculée (dans le cas des transitions de l'ensemble <*f*>, *s = fi(e)*). Les transitions <*e*>, <*e'*> et <*f'*> se produisent à chaque fois que la sortie est verrouillé (les gardes sont toutes à 0) et quand un changement sur l'une des gardes *gi* de la valeur 0 à 1 apparait : l'évaluation de la fonction *fi(e)* est nécessaire. Ici, on distingue deux cas: 1) si la valeur de la sortie *s* (représentée par la variable v_s), est égale à la valeur evaluée de la fonction *fi(e)*, alors on passe directement via les transitions de l'ensemble <*f'*> vers l'état stable *stable(s)*, 2) dans le cas contraire, on passe via les transitions de l'ensemble <*e*> (resp. <*e'*>) vers l'état de calcul *rising(s)* (resp. *falling(s)*) si l'évaluation de la fonction *fi(e)* est égale à 1 (resp. 0) afin d'attribuer la valeur *fi(e) = 1* (resp. *fi(e) = 0*) à la variable v_s et produire un front montant (resp. descendant) sur la sortie *s*, après l'écoulement d'un délai $\delta(s_{up})$ (resp. $\delta(s_{dn})$). Ce front produit sera propagé comme un évènement déclencheur pour les automates ultérieurs.
Les autres transitions réfèrent à l'occurrence des fronts sur les signaux apparus dans les affectations du processus et des changements sur les variables intermédiaires associées aux gardes *gi*, qui n'induisent pas de changements sur la valeur de la sortie s (les transitions qui bouclent).

2.3.3. Insertion des contraintes temporelles : dans ce modèle, pour chaque bloc dont la sortie est le signal *s*, les bornes des deux intervalles de délais, $l(s_{up})$, $u(s_{up})$, $l(s_{dn})$ ou $u(s_{dn})$ peuvent être instanciées ou bien laissées comme des paramètres (avec $l(s_{up}) \leq u(s_{up})$ et $l(s_{dn}) \leq u(s_{dn})$), selon le type de l'analyse subséquente effectuée (instanciée ou paramétrée). Chaque automate temporisé est déterministe dans les transitions d'actions, mais pas dans les transitions de temps (à moins que $l(s_{up}) = u(s_{up})$ et $l(s_{dn}) = u(s_{dn})$). Cependant, il arrive parfois que même dans le cas où toutes les bornes des intervalles sont toutes égales le modèle global est quasi-déterministe car des transitions concurrentes provenant des automates différents et franchissables au même instant peuvent provoquer de l'indéterminisme. Cependant, les sources d'indéterminisme sont très réduites.

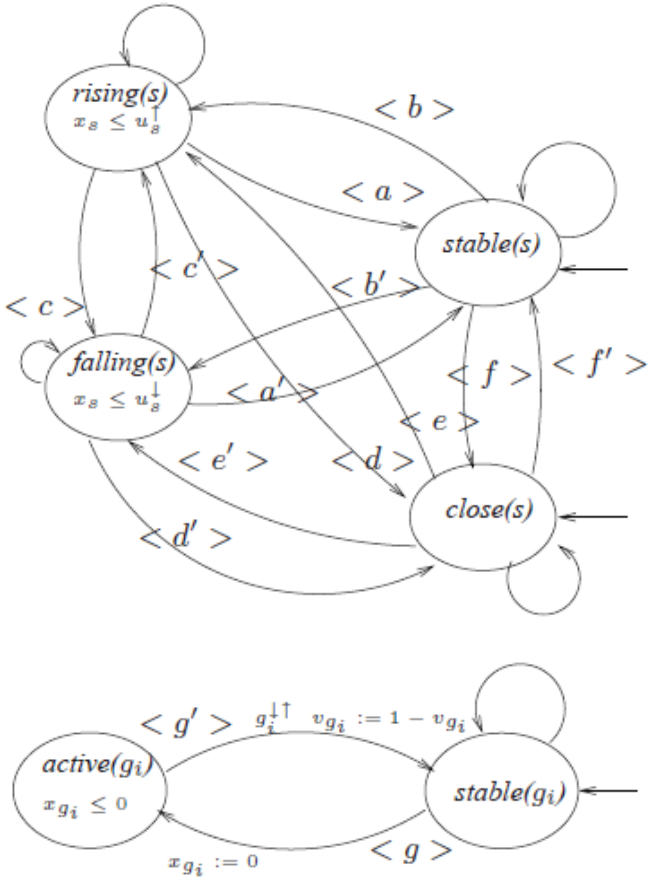


Figure 2: Les automates représentants les affectations des blocs séquentiels.

3. Implémentation (VHDL2TA)

Dans cette partie, on présente l'architecture de l'outil, ses modules, ses descriptions d'entrée/sortie et ses caractéristiques.

3.1. Architecture et implémentation.

Principalement, VHDL2TA intègre :

- un analyseur qui permet de donner une représentation structurelle aux descriptions d'entrée de l'outil (qui sont syntaxiquement correctes) : la description fonctionnelle du circuit en VHDL, la description temporelle des portes du circuit et la description d'environnement des signaux d'entrée.

- des modules de génération de descriptions d'entrée des outils d'analyse, UPPAAL, HyTech et IMITATOR-2, à partir du format intermédiaire produit par l'analyseur (l'utilisateur peut choisir le type de la description de sortie qu'on génère).

3.2. Les descriptions d'entrée et sortie.

Comme le montre la figure 4, l'outil prend en entrée les descriptions suivantes :

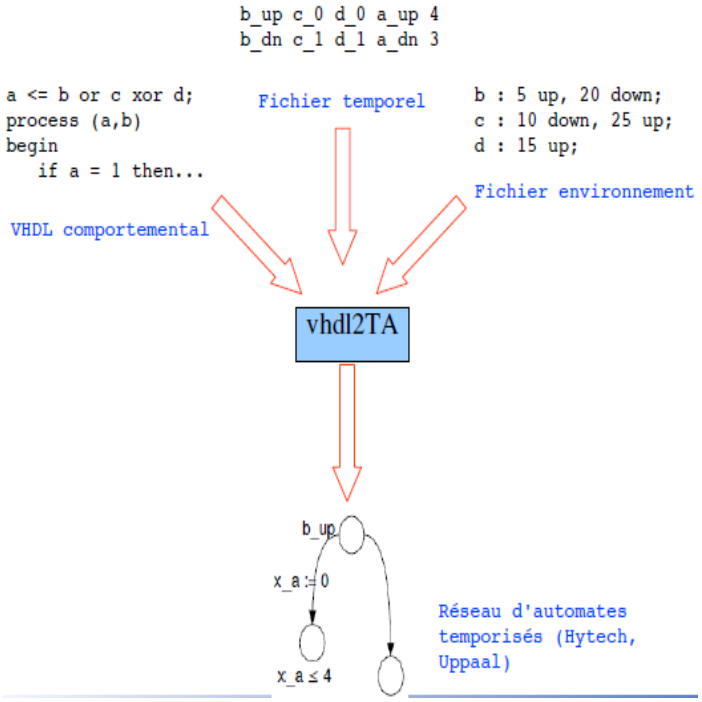


Figure 4: Les descriptions d'entrée / sortie de l'outil VHDL2TA.

la partie fonctionnelle du circuit (programme VHDL) est donnée par une description en VHDL. Chaque bloc combinatoire (resp. séquentiel) est décrit par une affectation concurrente (resp. processus).

la partie temporelle du circuit (délais des portes) décrit les délais de propagation des fronts sur la sortie des portes du circuit. Il existe deux types de descriptions : 1) les descriptions données sous forme d'un fichier dans lequel deux intervalles de délais sont associés à chaque porte (modèle temporel bi-bounded delay). 2) les descriptions dans lesquelles on associe un fichier de timing (configuration, front-sortie, temps) pour chaque porte (modèle STG complet). On leur ajoute aussi un fichier principal dans lequel on associe à chaque porte le nom (chemin) du fichier de timing correspondant. On note que dans ce modèle, les descriptions sont représentées par ces fichiers principaux.

L'environnement du circuit (Fichier d'environnement) décrit le comportement des signaux d'entrée.

L'outil VHDL2TA génère, ensuite, le modèle d'automates temporisés associé au circuit, en description d'entrée d'un outil d'analyse tel que UPPAAL, HyTech ou IMITATOR-2.

Dans certains cas, on passe aussi des spécifications en UPPAAL décrivant des propriétés temporelles du circuit à analyser, afin de les vérifier en employant le model-check d'UPPAAL. Les spécifications en UPPAAL sont décrites en logique temporelle TCTL. Les propriétés qu'on peut exprimer généralement dans les spécifications UPPAAL sont les propriétés d'accessibilité, de sureté et d'équité.

3.3. Fonctionnalités de l'outil

- Traduction des circuits décrits en VHDL en réseaux d'automates temporisés décrits dans le langage de description des outils d'analyse tels que : UPPAAL, HyTech, IMITATOR et IMITATOR-2.

- L'outil intègre un module, nommé figure2tv, qui permet de générer des programmes *VHDL* temporisé prêts à simuler à partir des descriptions d'entrée qu'on emploi pour générer les modèles d'automates temporisés. L'outil VHDL2TV [BE.10b] est dédié aussi pour cette tâche.

- L'outil intègre un module de réduction de figures (représentation structurelle du programme VHDL) produites lors de la phase de l'analyse syntaxique de description fonctionnelles des circuits. En conséquence, les modèles d'automates temporisés générés à partir des figures réduites, sont plus concis.

- En utilisant aussi le module de réduction des figures, on peut générer une version optimisée de la description fonctionnelle et de la description temporelle d'un circuit. A titre d'exemples, on a réussi à réduire l'architecture abstraite complète de la mémoire SPSMALL de 3 mots de 2 bits (resp. la mémoire SPSMALL de 1 mots de 2 bits), SPSMALL-3x2 (resp. SPSMALL-1x2), qui constitué de 92 blocs combinatoires et séquentiels (resp. 72 blocs) en 62 blocs (resp. 50 blocs).

- L'analyse instanciée sur les paramètres (simuler l'analyse paramétrée avec UPPAAL) pour réduire les temps de setup des signaux d'entrée du circuit, en utilisant le model-checker d'UPPAAL.

- Génération de fichiers de timing (de configurations – modèle temporel complet) des portes d'un circuit donné, à partir de sa description VHDL et du fichier de délais de ses portes (modèle temporel bi-bounded delay). Inversement, il existe aussi un moyen de générer un fichier des intervalles de délais des deux fronts pour chaque signal, à partir des fichiers de timing et de la description VHDL du circuit.

- Détection et élimination des fausses configurations dans les fichiers de timing associés aux portes d'un circuit donné.

4. Mode d'emploi de VHDL2TA

4.1. Installation.

L'outil est disponible en téléchargement sur le site web dont l'URL est donnée ci-dessous : http://www-asim.lip6.fr/~ema/valmem/vhdl2ta/index.html.

4.2. Les descriptions d'entrée de VHDL2TA

Dans cette partie, on définit le format des descriptions d'entrée de l'outil employées pour générer le modèle d'automates temporisés représentant un circuit donnée. On se sert de la porte *n_xor*

comme exemple, dont la sortie est nommée s et les deux entrées sont nommées e1 et e2, respectivement.

4.2.1. La description fonctionnelle du circuit est un programme VHDL, où les blocs combinatoires du circuit sont décrits par des affectations concurrentes et les blocs séquentiels sont décrits par des processus. Les programmes acceptés par l'analyseur de l'outil ne contiennent que des affectations simples et des processus de type If-elseif . . . -elseif {else} qui affectent une nouvelle valeur à un seul signal donné. Le format de ces affectations et de ces processus est décrit ci-dessous. Le fragment de la grammaire VHDL, qui génère ce sous ensemble du langage VHDL, est donné dans l'annexe B.1.

```
s <= f (e_1, . . . , e_m);

Process_Name Process (e_1, . . . , e_m)
begin
.         if guard_1
.                   then s <= f_1 () ;
.         else-if guard_2
.                   then s <= f_2 () ;
.         .
.         .
.         else-if guard_n
.                   then s <= f_n () ;
end
```

La description de la porte *n_xor* en VHDL est donné comme suit :

```
-- Entity Declaration
ENTITY n_xor IS
  PORT (
         s : out   BIT;
         e_1 : in    BIT;
         e_2 : in    BIT
  );
END n_xor;
```

```
-- Architecture Declaration
ARCHITECTURE arc OF n_xor IS

BEGIN
  s <= not (e_1 xor e_2);
END arc;
```

4.2.2. La partie temporelle du circuit (délais des portes) décrit les délais de propagation des fronts montants et descendants des signaux sur les portes du circuit. Généralement, la description est donnée sous forme d'un fichier dans lequel deux intervalles de délais (ou un délai ponctuelle) sont associés respectivement aux fronts montants et descendants de chaque porte (modèle *temporel bi-bounded delay*).

```
-- Delai pour chaque front
  s : [1,2]- , [3;4]+ ;
```

Une autre option consiste à associer un fichier de timing pour chaque porte (modèle STG complet). Dans ces fichiers, on associe un délai ponctuel pour chaque configuration et pour chaque front d'un signal d'entrée d'une porte qui induit un changement sur la sortie. Un exemple d'un modèle temporel complet pour la porte *n_xor* est donné ci-dessous. On peut remarquer que le premier modèle (modèle *temporel bi-bounded delay*) est une abstraction de ce dernier modèle.

```
-- Delai pour chaque configuration
  e_1 e_2 s time
  0  u  d  2ns
  0  d  u  3ns
  u  0  d  2ns
  d  0  u  3ns
  1  d  d  1ns
  1  u  u  4ns
  d  1  d  1ns
  u  1  u  4ns
```

Dans ce modèle, une phase d'abstraction de délais (introduite dans l'outil) est nécessaire pour extraire pour chaque porte du circuit deux intervalles de délais (abstractions de délais) associés aux fronts montants et descendants propagés sur la sortie de la porte, à partir de son fichier de timing.

4.2.3. La description d'environnement est donnée sous forme d'un fichier qui contient une liste de comportements associés aux signaux d'entrée du circuit à modéliser. Un comportement d'un signal est une suite alternée de fronts montants et descendants à des instants croissants (ou intervalle d'instants). La syntaxe des descriptions d'environnement est décrite par une grammaire génératrice donnée dans l'annexe B.3.

Voici un exemple de description d'environnement des signaux d'entrée de la porte *n_xor* :

```
-- Env
e_1 : 5 up, 15 down;
e_2 : 10 up, 20 down;
```

4.2.4. Fichier des propriétés TCTL contient les comportements des signaux de sortie qui sont exprimés en propriétés de la logique TCTL afin d'effectuer le model-checking avec UPPAAL. Le but est de vérifier si le modèle d'automates généré satisfait ces propriétés.

Dans l'exemple de la porte *n_xor*, le comportement de la sortie de la porte qu'on attend :

```
s : [6,7] dn, [13,14] up, [16,17] dn, [23,24] up ;
```

est décrit par la propriété TCTL présentée ci-dessous.

```
 A[] ((((t>=0 and t<6) or (t>14 and t<16) or t>24) imply s==1) and (((t>7 and
t<13) or (t>17 and t<23)) imply s==0))
```

La BNF des propriétés de la logique TCTL employées dans les spécifications UPPAAL est présentée dans l'annexe B.4.

4.3. Les commandes d'appel de VHDL2TA

VHDL2TA peut être employé en plusieurs modes différents :

- *Modèle d'automates temporisés* : on génère la description de réseaux d'automates temporisés à partir de la description comportementale en VHDL, de la description temporelle du circuit et du comportement des signaux d'entrée.
- *Programme VHDL temporisé prêt à simuler* : on génère ce dernier programme à partir des mêmes descriptions d'entrée que celles employées dans le mode précédent.
- *Modèle temporel* : il existe deux sous modes principaux. Dans la premier mode (*modèle temporel complet*), on génère les fichiers de timings associés aux portes du circuit, à partir de leur fonctionnalité et les intervalles de délais de propagation de ses fronts. Dans le deuxième mode (*modèle temporel bi-bounded delay*), on génère un fichier d'intervalles de délais des deux fronts pour chaque porte à partir des fichiers de timing et de la description en VHDL du circuit.
- *Optimisation VHDL :* ce mode permet de générer une description fonctionnelle et une description temporelle optimisées à partir de la description en VHDL initiale et de la description temporelle initiale.

4.3.1. Modèle d'automates temporisés

Ce mode permet de générer le modèle d'automates temporisés en descriptions d'entrée des outils tels que, UPPAAL, Hytech ou IMITATOR-2, à partir de la description fonctionnelle en VHDL et éventuellement de la description temporelle du circuit et de l'environnement des signaux d'entrée du circuit. La syntaxe de la commande qu'on passe sur l'outil est donnée comme suit.

```
vhdl2ta <vhld_file> [-env <env_file>] [-temp <temp_file> temp_type] [-mode ta] [-out <ta_file>] [ta_options] [options]
```

Les paramètres passés dans cette commande :

`vhdl_file` : le fichier VHDL qui décrit la fonctionnalité du circuit. Ce paramètre est toujours obligatoire.

`-temp` (abrégé à `-t`) : c'est l'option qui permet de décrire la partie temporelle du circuit. Elle prend en paramètre, le fichier temporel `temp_file` et son type `temp_type` (1 s'il s'agit d'un modèle temporel complet, 2 s'il s'agit d'un modèle *temporel bi-bounded delay*). Par défaut, les délais des portes sont nuls, si cette option n'est pas spécifiée. On peut mettre `-t` à la place de `-temp` dans la commande.

`-env` (abrégé à `-e`) : cette option décrit l'environnement du circuit. Elle permet de fournir le fichier d'environnement `env_file`. Par défaut, le comportement des signaux d'entrée du circuit est stable à la valeur 0, si cette option n'est pas passée en paramètres de la commande `vhdl2ta`.

`-mode` (abrégé à `-m`) : mode d'emploi de VHDL2TA. Dans ce mode d'emploi, il prend la valeur `ta`. On note que cette option prend par défaut cette dernière valeur. Donc, il n'est pas obligatoire de la spécifier dans ce mode. Voici les valeurs que l'option `-mode` prend dans tous les modes d'emploi de VHDL :

- `ta` : modèle d'automates temporisés (voir section 4.3.1).
- `sim` : programme VHDL temporisé prêt à simuler (voir section 4.3.2).
- `temp` : modèle temporelle (voir section 4.3.3)
- `vhdl_opt` : optimisation VHDL (voir section 4.3.4)

`-out` (abrégé à `-o`) : cette option permet de passer en paramètre le nom de la description de sortie. Par défaut, le nom de la description prend le même nom que celle de la description en VHDL en changeant l'extension *.vhd* par l'extension de la description de sortie (ex: Hytech *.hy*, UPPAAL, *.ta*, . . . , etc).

Les autres options (`ta_options`) spécifiques pour ce mode sont :

`-ta_format` (abrégé à `-taf`) : indique le format de la description à générer. Les paramètres que l'option peut prendre sont :

- `hy` : s'il s'agit d'une description en format HyTech.
- `upl` : s'il s'agit d'une description en format UPPAAL.
- `imi` : s'il s'agit d'une description en format IMITATOR.
- `imi2` : s'il s'agit d'une description en format IMITATOR-2.

L'option `-ta_format` prend aussi un deuxième paramètre optionnel quand le premier paramètre passé est égal à `imi2`. Ce paramètre indique le mode dans lequel les descriptions en IMITATOR-2 sont générées. On note que l'outil IMITATOR-2 accepte trois modes d'emploi : **1)** `reachability` qui permet de faire l'analyse d'accessibilité 2) `inversemethod` qui permet d'appliquer d'algorithme de la méthode inverse [ACEF.09] pour la synthèse des contraintes sur les paramètres 3) `bca` qui permet d'appliquer l'algorithme BCA (Behavioral Cartography Algorithm [AF.10]) afin de calculer la cartographie du système avec les deux variantes `cover` et `randomX`

`-ta_version` (abrégé à `-tav`) : indique le type du modèle d'automates temporisés à générer. On distingue deux types de modèles : **1)** le modèle original, dans lequel on associe des locations (X0_i, X1_i, F_i) pour chaque affectation concurrente (i) d'un processus branché par une garde. **2)** le modèle dans lequel on regroupe toutes les locations associées aux affectations concurrentes (i) d'un processus en locations X0, X1 et F. C'est le modèle d'automates présenté dans la section 2 de ce manuel et qui a été employé dans les tests des circuits présentés dans [BE.11a] (la page web de l'outil VHDL2TA). Ce dernier modèle est une abstraction du premier. Par défaut, cette option prend la valeur `model_grp_locs` représentant le deuxième modèle (le premier modèle est représenté par la valeur `model_ecl_locs`).

`-param_type` (abrégé à `-pt`) : cette option permet d'indiquer le type des paramètres temporels à employer. Il existe deux types : 1) associer un seul paramètre pour un évènement temporel (ex: delta0, délai de propagation des fronts descendants sur la sortie de la porte n_xor). 2) associer deux paramètres pour un évènement temporel représentant les deux bornes extrêmes de ce dernier (ex: [delta0_l, delta0_u] représente l'intervalle de délais de propagation des fronts descendants sur la sortie de la porte n_xor). Par défaut, l'option prend cette dernière valeur. Ce dernier type est bien adapté pour représenter les intervalles de délais de propagation des fronts sur la sortie des portes de circuits. Bien que, on peut spécifier un intervalle de délais avec un seul paramètre dans les descriptions en HyTech et IMITATOR avec un seul paramètre, contrairement aux descriptions en UPPAAL (ne traite pas le cas paramétré).

-fig_opt (abrégé à -fo) : cette option est employée uniquement dans le mode d'emploi ta du fait de son utilité. Elle permet d'indiquer à l'outil VHDL2TA d'optimiser la figure représentant la description comportementale en VHDL, avant de générer le modèle d'automates temporisés. Ceci permet d'avoir des modèles plus optimisés que le modèle original. A titre d'exemple, on n'a réussi à passer un modèle d'un circuit sur l'outil d'analyse HyTech qu'après l'utilisation de cette option. Bien que le modèle de ce circuit passe bien sur les autres outils d'analyse sans aucune optimisation de figure.

-parametric_analysis (abrégé à -pa) : cette option est n'employée que dans le mode d'emploi ta et quand l'option -ta_format est égale au format upl (descriptions en UPPAAL). En utilisant cette option, on peut effectuer l'analyse instanciée sur les paramètres temporels, tels que les temps de setup et de hold des signaux d'entrée des circuits analysés en utilisant le model-checker d'UPPAAL afin de réduire leurs valeurs. Cette option prend en paramètres : 1) le fichier des propriétés TCTL (obligatoire) qui décrit ce qu'on veut vérifier 2) le type de paramètres à réduire t_setup s'il s'agit des temps de setup et t_hold s'il s'agit des temps de hold (paramètre optionnel, par défaut c'est t_setup). On note que dans la plupart du temps, ces propriétés décrivent le comportement des signaux de sorties des circuits analysés. La description en UPPAAL générée intègre toutes les valeurs optimales des temps de setup ou des temps de hold réduits.

D'autres options ([options]) sont possibles à passer en paramètres pour la commande vhdl2ta. Ces options sont employées dans tous les modes d'emploi de VHDL2TA. On va les découvrir plus loin dans la section 4.3.5.

4.3.2. Programme VHDL temporisé prêt à simuler

Dans ce mode, ce programme est généré à partir des mêmes descriptions d'entrée que celles employées dans le mode précédent. La syntaxe de la commande est presque le même que celui du mode précédent :

vhdl2ta <vhld_file> [-env env_file] [-temp <temp_file> temp_file] **-mode sim** [-out testbench_file] [sim_options] [options]

Comme on peut le voir, la plupart des paramètres passés dans cette commande sont décrits précédemment. Il reste à savoir que dans ce mode on passe en paramètre de l'option -out le nom du fichier testbench qui constitue, avec les autres fichiers VHDL décrivant la partie fonctionnelle et temporelle des portes du circuit, le programme prêt à simuler. Tous les fichiers *.vhd* générés sont dans le même répertoire courant que celui du fichier testbench.

Les autres options (ta_sim) spécifiques pour ce mode sont :

-sim_version (abrégé à -simv) : il existe deux versions des programmes VHDL prêts à simuler qu'on peut générer avec l'outil VHDL2TA. **1)** la première version dans laquelle on associe un seul composant pour tout le circuit (le programme n'est constitué que deux fichiers). **2)** la deuxième version dans laquelle un composant VHDL est associé pour chaque porte du circuit en décrivant sa fonctionnalité et ses annotations temporelles (en conséquence, le programme est constitué d'un fichier testbench et de plusieurs fichiers VHDL). Par défaut, l'option prend la valeur gate

représentant la deuxième version. A titre d'indication, la première version est représentée par la valeur `circuit`.
`-sim_format` (abrégé à `-simf`) : indique le format VHDL dans lequel les composants associés aux portes du circuit sont représentés. Les composants peuvent être représentés soit par des processus (`process`) soit par des affectations concurrentes (`assign`). Il est très recommandé d'employer la première représentation pour différents raisons car elle est bien adaptée pour représenter les blocs séquentiels et elle permet une meilleure lisibilité du code. Cette option prend la première valeur, par défaut.

4.3.3. Modèle temporel

Ce mode contient aussi deux sous modes principaux :

modèle temporel complet : ce mode permet de générer les fichiers de timing associés aux portes du circuit à partir de la description fonctionnelle du circuit en VHDL et de la description temporelle donnée sous forme d'un fichier d'intervalles de délais. La syntaxe de la commande employée ici est donnée comme suit.

```
vhdl2ta <vhld_file> -temp <temp_file> 2 -mode temp [-out <temp_file>] [options]
```

modèle temporel bi-bounded delay : Inversement, en utilisant la syntaxe de la commande présentée ci-dessous, on peut générer un fichier d'intervalles de délais à partir des fichiers de timing associés aux portes du circuit ainsi que sa description fonctionnelle.

```
vhdl2ta <vhld_file> -temp <temp_file> 1 -mode temp [-out <temp_file>] [options]
```

Il existe aussi un autre mode qui permet de détecter et d'éliminer les fausses configurations dans les fichiers de timing associés aux portes de circuits. Dans ce mode, il est indispensable d'indiquer le mode `configs_check` qu'on passe en paramètre dans l'option `-temp_mode`. La syntaxe de la commande qu'on passe dans ce mode est donnée ci-dessous. On note que dans les deux modes précédents, l'option `-temp_mode` prend en paramètres les deux valeurs `timing_files` et `delay_file`, respectivement.

```
vhdl2ta <vhld_file> -temp <temp_file> 2 -mode temp [-out <temp_checked_file>] -
temp_mode  configs_check [options]
```

Dans le premier mode, on peut employer l'option `-temp_select` (abrégé à `-ts`) avec laquelle on peut spécifier les critères de sélection des délais des configurations sur un intervalle de délais. Les critères de sélection que cette option peut prendre en paramètre sont listés ci-dessous :

- `min` : sélectionner les bornes inférieures des intervalles des délais associées aux portes.
- `max` : sélectionner les bornes supérieures.
- `moy` : sélectionner les valeurs moyens (leur partie entière supérieure) des bornes des intervalles.
- `random` : sélectionner des délais aléatoire dans les intervalles des délais (c'est la valeur que `temp_select` prend par défaut).

On note que cette option peut être employée dans le mode d'emploi de l'outil VHDL2TA `sim`, quand l'option `-temp` prend des fichiers temporelles de type 2 (fichiers de délais).

D'autre part, pour avoir des fichiers de timing avec des délais nuls pour toutes les configurations, il suffit d'omettre l'option -temp. On note que l'option -temp_mode (abrégé à -tm) prend par défaut la valeur delay_file si le type du fichier passé dans l'option -temp est égal à 1, la valeur timing_files si le type est égal à 2.

4.3.4. Optimisation VHDL

Dans ce mode, l'outil VHDL2TA prend en entrée la description comportementale en VHDL du circuit à optimiser et éventuellement sa description temporelle sous forme d'un fichier d'intervalle de délais, ensuite il génère la version optimisée de ses descriptions. La syntaxe de la commande employée dans ce mode est donnée comme suit :

```
vhdl2ta <vhld_file> [-temp temp_file temp_type] -mode vhdl_opt [-out <vhdl_file> [<temp_file>]] [options]
```

Comme on peut le constater, cette commande peut prendre en paramètres les noms des deux descriptions fonctionnelles et temporelles simultanément.

4.3.5. Options générales

On distingue deux classes d'options : des options spécifiques à un mode donné (qu'on vient de voir ci-dessus); et des options générales qui peuvent être employées dans tous les modes. Ces dernières sont données comme suit.

-mode (**par défaut :** ta) : comme c'est déjà indiqué auparavant, c'est le mode d'emploi de VHDL2TA.

-debug (abrégé à -d) : cette option permet de donner des informations de debogage.

-warning (abrégé à -w) : cette option permet d'afficher les warnings.

4.3.6. Exemples d'Appels

La commande vhdl2ta n_xor.vhd -e n_xor.env -t n_xor.tmp 2 -m ta -o n_xor.ta -taf upl génère le modèle d'automates temporisés (associé à la porte *n_xor*) décrit en format UPPAAL dans le fichier n_xor.ta. Cette description en UPPAAL est décrite dans l'annexe A.1.

La commande vhdl2ta d_flip_flop.vhd -e d_flip_flop.env -t d_flip_flop.tmp 2 -m ta -o d_flip_flop.imi -taf imi2 inversemethod génère le modèle d'automates temporisés décrit dans le fichier *.imi* et une valuation de référence *Pi0* décrite dans le fichier de référence *.pi0*. Avec ces deux fichiers, on peut appeler IMITATOR-2 en mode inversemethod (la méthode inverse) afin de générer le système de contraintes linéaires sur les paramètres de délais dont les valuations assurent le même fonctionnement que celui de la valuation de référence *Pi0*.

La commande vhdl2ta sp_3x2.vhd -e sp_3x2.env -t sp_3x2.tmp 2 -m ta -o sp_3x2_opt.ta -taf upl -fo génère une description en UPPAAL à partir de la figure optimisée. La description

générée contient 30 automates de moins que la description en UPPAL générée sans utiliser cette option. Donc, avec cette option on obtient des modèles optimisés.

La commande `vhdl2ta sp_3x2.vhd -t sp_3x2.tmp 2 -m vhdl_opt -o sp_3x2_vhld_opt.vhd` génère une version optimisée d'une description fonctionnelle et temporelle de l'architecture abstraite de la mémoire SPSMALL. En utilisant ce mode, cette dernière est réduite de 92 à 62 portes.

La commande `vhdl2ta lsv1.vhd -e lsv1.env -t lsv1.tmp 2 -m ta -o lsv1.ta -taf upl -pa lsv1.q t_setup` permet d'effectuer l'analyse instanciée sur les paramètres en utilisant le model-checker d'UPPAAL afin de réduire les temps de setup des signaux d'entrée du circuit décrit `lsv1`. Les propriétés à vérifier sont décrites dans le fichier `lsv1.q`. La description en UPPAAL générée `lsv1.ta` intègre toutes les valeurs optimales des temps de setups réduits.

La commande `vhdl2ta sp_3x2.vhd -t sp_3x2.tmp 1 -m temp -o sp_3x2_check.tmp -tm configs_check` permet de détecter et d'éliminer les fausses configurations dans les fichiers de timing associés aux portes de l'architecture abstraite de la mémoire SPSMALL. Le fichier principal `sp_3x2_check.tmp` contient les chemins vers les nouveaux fichiers de timing qui ne contiennent que les bonnes configurations.

La commande `vhdl2ta sp_3x2.vhd -t sp_3x2.tmp 1 -m temp -o sp_3x2_delays.tmp -tm delay_files` permet de générer le fichier d'intervalles de délais à partir des fichiers de timing associés aux portes du circuit et aussi à partir de la description fonctionnelle `sp_3x2.vhd`.

La commande `vhdl2ta sp_3x2.vhd -t sp_3x2_timing.tmp 2 -m temp -o sp_3x2_timing.tmp -tm timing_file -ts random` permet de générer les fichiers de timing associés aux portes de l'architecture abstraite de la mémoire SPSMALL à partir de la description fonctionnelle du circuit en VHDL `sp_3x2.vhd` et à partir du fichier de délais `sp_3x2_delays.tmp`. Les délais des configurations sur les intervalles de délais sont sélectionnés d'une manière aléatoire.

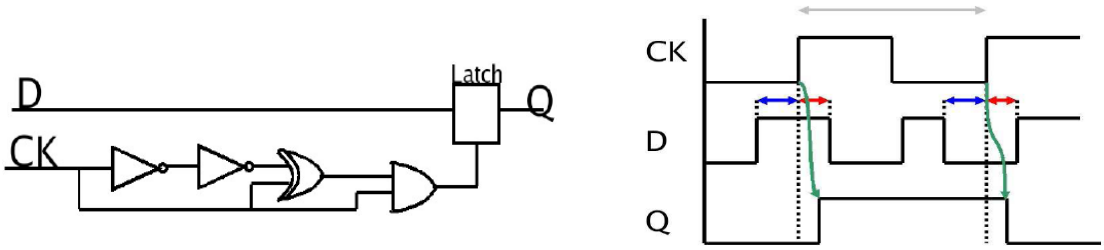
La commande `vhdl2ta n_xor.vhd -e n_xor.env -t n_xor_timing.tmp 1 -m sim -o n_xor_testbench.vhd` génère le programme VHDL prêt à simuler dont le fichier testbench est nommé `n_xor_testbench.vhd`.

5. Exemple de tests

Dans cette section on présente un exemple de tests pour l'outil VHDL2TA. L'intégralité des tests déroulés sur divers circuits est synthétisée dans [BE.11b].

Circuit Valmem latch:

Dans cette partie, nous effectuons quelques tests sur le circuit latch étudié dans le cadre du projet VALMEM [AEF.09] [AEF.08]. Le circuit et la description des paramètres de sa spécification (les temps de setup T*setup* et de hold *Thold* du signal d'entrée *D*, la durée du front montant *THI* de l'horloge *CK*, la durée du front descendant T*LO* de l'horloge *CK* et le temps de réponse T*CK->Q*) sont donnés sur la figure 5 (extraite de [AEF.08]).



(a) (b)

Figure 5 : a) Le circuit valmem latch. b) la spécification du circuit.

L'implémentation du circuit en VHDL est donnée comme suit :

```
ENTITY valmem_latch IS
  PORT (
         q          : out    BIT;
         ck         : in     BIT;
         d          : in     BIT
  );
END valmem_latch;

ARCHITECTURE RTL OF valmem_latch IS
  SIGNAL not1 : BIT;
  SIGNAL not2 : BIT;
  SIGNAL xor1 : BIT;
  SIGNAL and1 : BIT;
BEGIN
 not1 <= not ck;
  not2 <= not not1;
  xor1 <= not2 xor ck;
  and1 <= xor1 and ck;
  latch: process (and1, d)
  begin
    if and1 = '1' then
      q <= d;
    end if;
  end process;
END ;
```

Nous avons effectué deux tests similaires.

Premier test : Nous avons pris un environnement de signaux d'entrées et avons fixé des délais pour les portes du circuit avec lesquels la spécification mentionnée ci-dessous est satisfaite. Puis, nous avons montré que le comportement du signal de sortie *Q* est conforme à ce qui est décrit dans la spécification. Le temps de réponse *TCK->Q* doit être inférieur au cycle de l'horloge. Nous avons effectué l'analyse en utilisant le model-checker UPPAAL sur le modèle généré par l'outil VHDL2TA.

La description de ces délais des portes et la description de cette environnement sont donnés comme suit :

```
Env1:
automaton environment;
clock ck with thi 1000 tlo 1000 inverse ncycles 2;
--ck  : 1000 up, 2000 down, 1000 up, 2000 down;
d : 1000 up, 1634 down, 2200 up, 3000 down, 3634;

Délais1 :
not1 : 147-, 219+ ;
not2 : 163-, 155+ ;
xor1 : 416-, 147+ ;
and1 : 155-, 80+ ;
latch : 240-, 240+ ;
```

La description en UPPAAL générée par l'outil est caractérisée par :

- 5 automates associés aux signaux de sortie et aux signaux internes de circuit + les automates d'environnement.
- 5 horloges locales correspondant aux automates + l'horloge globale.
- 2+5 variables discrètes associées aux signaux du circuit.
- 4 × 5 paramètres associés aux délais des fronts montants et descendants des signaux du circuit.
- 3 × 4 + 4 × 1 locations des automates associés aux portes du circuit + les locations associées aux automates d'environnement.

La commande employée pour générer cette description est mentionnée ci-dessous :

```
vhdl2ta vlm_latch.vhd -e vlm_latch.env -t vlm_latch.tmp 2 -m ta -o vlm_latch.ta
-taf upl
```

La propriété TCTL qui décrit le comportement du signal Q attendu suivant :

```
Q : 1467 up, 3467 down;
```

est donnée comme suit :

```
A[]((((t>=0 and t<1467) or t>3467) imply q==0) and (t>1467 and t<3467 imply
q==1))
```

Cette propriété TCTL est bien vérifiée par le modèle d'automates temporisés décrit en Uppaaal. Donc, le temps de réponse est bien conforme à celui décrit dans la spécification.

Nous notons que le temps de la vérification de cette propriété est inférieur à 0,1 sec; et le temps de génération de la description en UPPAAL est inférieur à 0,1 sec.

Comme on peut le constater dans ce test, la valuation (point) *Pi0* associée aux valeurs des paramètres de la spécification est donnée comme suit:

$$\begin{array}{llll} T_{HI} = 1000 & T_{LO} = 1000 & T_{Hold} = 350 & T_{Setup} = 0 \\ \delta_{Not1\uparrow} = 219 & \delta_{Not1\downarrow} = 147 & \delta_{Not2\uparrow} = 155 & \delta_{Not2\downarrow} = 163 \\ \delta_{Xor\uparrow} = 147 & \delta_{Xor\downarrow} = 416 & \delta_{And\uparrow} = 80 & \delta_{And\downarrow} = 155 \\ \delta_{Latch\uparrow} = 240 & & & \end{array}$$

Deuxième test: Nous avons refait un test identique au précédent mais cette fois avec un environnement plus simple qui satisfait les valeurs des paramètres des signaux d'entrée du point *Pi0*. L'environnement est décrit comme suit :

E*nv1:*

```
automata environment;
clock ck with thi 1000 tlo 1000 inverse ncycles 1;
--ck  : 1000 up, 2000 down down;
d : 1000 up, 1634 down;
```

Le comportement de la sortie *Q* attendu est donné ci-dessous. Le temps de réponse est bien conforme à ce qui était spécifié.

```
Q : 1467 up;
```

Nous avons fait encore des tests pour toutes les valeurs de *Thold* $\in$ *{634, . . . , 717}*. Le temps de réponse *TCK->Q* est compris dans l'intervalle *]633,718[* (le comportement du signal de sortie *Q* est toujours le même). Donc, la spécification est toujours satisfaite.

Nous avons repoussé, ensuite, les tests cette fois en utilisant la description en IMITATOR-2 générée par l'outil VHDL2TA. Nous avons attribué des modifications sur cette description en remplaçant manuellement les valeurs de temps de setup et de hold du signal *d* par des paramètres. La description obtenue est employée pour générer le système de contraintes synthétisées pour ce point en utilisant l'outil IMITATOR-2 (méthode inverse). Ce système de contraintes synthétisées qu'on note par *K* est exactement le même que celui présenté dans [AEF.09]. Il est donné comme suit.

$$\begin{array}{rcl} T_{LO} & \geq & 0 \\ T_{HI} & > & T_{Hold} + \delta_{Latch\downarrow} \\ T_{Hold} + \delta_{Latch\downarrow} & > & \delta_{Not1\downarrow} + \delta_{Not2\uparrow} + \delta_{Xor\downarrow} + \delta_{And\downarrow} \\ T_{Hold} & > & \delta_{Not1\downarrow} + \delta_{And\uparrow} + \delta_{Latch\uparrow} \\ \delta_{Not1\downarrow} + \delta_{Not2\uparrow} + \delta_{Xor\downarrow} & > & T_{Hold} \\ T_{Setup} & = & 0 \\ \delta_{And\uparrow} + \delta_{Latch\uparrow} & > & \delta_{Not2\uparrow} \\ \delta_{Not1\downarrow} & = & \delta_{Xor\uparrow} \\ \delta_{Not2\uparrow} & > & \delta_{And\uparrow} \\ \delta_{And\uparrow} & > & 0 \\ \delta_{Not1\uparrow} & > & \delta_{Not1\downarrow} \\ \delta_{Not1\downarrow} & > & 0 \end{array}$$

Nous avons aussi confirmé que les valeurs de temps de thold est compris dans l'intervalle *]633,718[*, en utilisant aussi la méthode inverse. En fait, nous avons re-généré le système de contraintes à nouveau, en remplaçant manuellement cette fois que les valeurs de temps de hold par des paramètres dans la description en IMITATOR-2 générée. Tous les autres paramètres sont instanciés par les valeurs du point *Pi0*. Les contraintes obtenues sur le temps de thold sont données par $633 < thold < 718$.

6. Remarques finales

L'outil VHDL2TA permet de générer des modèles d'automates temporisés en les décrivant dans le format d'entrée des outils d'analyse UPPAAL, HyTech ou IMITATOR (1 et 2). Ces modèles représentent le comportement fonctionnel et temporel de circuits à analyser. L'outil est testé sur divers exemples de circuits asynchrones (certains avec des mémoires). De plus, les modèles générés représentant des circuits à cent portes ont été analysés par le model-checker d'UPPAAL. C'est le cas de l'architecture complète abstraite de la mémoire SPSMALL de 3 mots de 2 bits, extraite au LIP6. Une présentation détaillée de l'analyse de cette dernière est donnée dans [BE.10a].

L'outil VHDL2TA a aussi été employé dans l'approche d'analyse de circuits proposée dans [BBCELR.10]. Cette approche permet d'offrir une nouvelle opportunité aux concepteurs de circuits. Ils pourraient analyser automatiquement la fonctionnalité et les propriétés temporelles de leurs circuits en comblant l'écart entre les modèles de circuits (conçus et décrits en VHDL) et l'efficacité des outils de model-checking temporisés. Cette approche peut être aussi employée pour accélérer la simulation temporelle des circuits complexes et la rendre plus précise en partant de la description au un niveau transistor et de la simulation électrique pour passer à un niveau de simulation RTL. Le modèle au niveau transistor de chaque porte est nécessaire pour caractériser les délais appropriés. Une fois que le délai de chaque porte est extrait, la simulation de tout le système est effectuée au niveau logique.

Références

[A.09] E. André. IMITATOR: A Tool for Synthesizing Constraints on Timing Bounds of Timed Automata. In Martin Leucker and Carroll Morgan (eds.), ICTAC'09, LNCS 5684, pages 336-342. Springer, 2009.

[A.10] E. André. IMITATOR II: A Tool for Solving the Good Parameters Problem in Timed Automata. In Yu-Fang Chen and Ahmed Rezine (eds.), INFINITY'10, Electronic Proceedings in Theoretical Computer Science 39, pages 91-99. 2010.

[AD.94] R. Alur and D. L. Dill. A Theory of Timed Automata. TCS, 126(2):183–235, 1994.

[ACEF.09] E. André, T. Chatain, E. Encrenaz, L. Fribourg, An inverse method for parametric timed automata, International Journal of foundations of Computer Sciences, vol 20, no 5, pages 819-836, World Scientific Publishing Company.

[AEF.08] E. André, E. Encrenaz, L. Fribourg, Modèles d'automates temporisés pour l'analyse de circuits mémoire. Délivrable D3.1 - Projet VALMEM, Janvier 2008.

[AEF.09] E. André, E. Encrenaz, L. Fribourg, Synthesizing Parametric Constraints on Various Case Studies Using IMITATOR. Research Report LSV-09-13, Laboratoire Spécification et Vérification, ENS Cachan, France, June 2009. 18 pages.

[AF.10] E. André et L. Fribourg. Behavioral Cartography of Timed Automata. In Antonin Kucera and Igor Potapov (éditeurs), RP'10, LNCS 6227, Springer, pages 76–90, septembre 2010.

[BBCELR.10] A. Bara, P. Bazargan-Sabet, R. Chevallier, E. Encrenaz, D. LeDu, P. Renault. Formal Verification of Timed VHDL Programs. International Forum on Specification and Design Languages, 2010.

[BE.10a] A. Bara, E. Encrenaz. Analyse de l'architecture abstraite de la mémoire SP-SMALL (3x2 bits), (Reunion VALMEM), Dec 2010.

[BE.10b] A. Bara, E. Encrenaz. VHDL2TV : outil de génération des programmes VHDL temporisés prêts à simuler, rapport interne de l'equipe Soc – Lip6, 2010.

[BE.11a] A. Bara, E. Encrenaz. VHDL2TA: http://www.lsv.ens-cachan.fr/~encrenaz/valmem/vhdl2ta/index.html, 2011.

[BE.11b] A. Bara, E. Encrenaz. VHDL2TA : outil de traduction des circuits décris en VHDL en automates temporisés – Jeu de Tests, rapport interne de l'equipe Soc – Lip6, 2011.

[BJMY.02] M. Bozga, H. Jianmin, O. Maler, and S. Yovine. Verification of asynchronous circuits using timed automata. In TPTS'02, ENTCS, volume 65, 2002.

[CEFX.06] R. Chevallier, E. Encrenaz-Tiphene, L. Fribourg, W. Xu, *Timing Analysis of an Embedded Memory: SPSMALL*, WSEAS Transactions on Circuits and Systems, vol 5(7), pp 973-978, 2006.

[HHW.95] T. A. Henzinger, P. Ho, and H. Wong-Toi. A user guide to HyTech. In TACAS, pages 41–71, 1995.

[LBG.98] Lester Anthony, Bazargan Sabet Pirouz, and Greiner Alain. Yagle, a second generation functional abstractor for cmos vlsi circuits. In 10th International Conference on Microelectronics, pages 265–268, Monastir, Tunisia, December 1998.

[LPY.97] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. International Journal on Software Tools for Technology Transfer, 1:134–152, 1997.

[MP.95] O. Maler and A. Pnueli. Timing analysis of asynchronous circuits using timed automata. In Int. conf. on Correct Hardware Design and Verification Methods (CHARME), volume 987, pages 189–205. Springer, 1995.

[Y.97] S. Yovine. Kronos: A Verification Tool for Real-Time Systems. International Journal on Software Tools for Technology Transfer, 1:123–134, 1997.

A. Exemples de circuits

A.1. Le modèle d'automates temporisés décrit en UPPAAL du circuit n_xor

```
/*----------------------------- Description du systeme -----------------------------*
 *----------------------------------------------------------------------------------*/

/*-- Definition des variables globales du systeme: horloges, discretes, paramètres
 *-------------------------------------------------------------------------------------------------*/

clock       t,
            x_s;

int         e_1 := 0,
            e_2 := 0,
            s := 1;


const int   delta0_l_s = 1, delta0_u_s = 2, delta1_l_s = 3, delta1_u_s = 4 ;

broadcast chan          e_1_down, e_1_up,
                        e_2_down, e_2_up,
                        s_down, s_up;

/*--  les automates de l'environnement qui définissent les signaux d'entrée --*
 *-----------------------------------------------------------------------------*/
process env_nd_signals ()
{
            state       env_nd_signals_0 {t <= 5},
                        env_nd_signals_1 {t <= 10},
                        env_nd_signals_2 {t <= 15},
                        env_nd_signals_3 {t <= 20},
                        env_nd_signals_end {true};
            init        env_nd_signals_0;
            trans
    env_nd_signals_0 -> env_nd_signals_1 { guard t == 5; sync e_1_up!; assign e_1:=1; },
    env_nd_signals_1 -> env_nd_signals_2 { guard t == 10; sync e_2_up!; assign e_2:=1; },
    env_nd_signals_2 -> env_nd_signals_3 { guard t == 15; sync e_1_down!; assign e_1:=0; },
    env_nd_signals_3 -> env_nd_signals_end { guard t == 20; sync e_2_down!; assign e_2:=0; } ;
}

/*-- les automates qui définissent tous les signaux auxiliaires et les signaux de sortie --*
 *---------------------------------------------------------------------------------------*
*/

//-- les automates qui définissent le signal s: //
//---------------------------------------- ------//
process Ass_s ()
{
            state       l_x0_s {x_s <= delta0_u_s}, l_x1_s {x_s <= delta1_u_s}, l_f_s {true};
            init        l_f_s ;
            trans
l_x0_s -> l_x0_s        { guard e_1 == 0; sync e_1_down?; },
l_x0_s -> l_x0_s        { guard e_1 == 1; sync e_1_up?; },
l_x0_s -> l_x1_s        { guard e_1 == 1-0   and e_2 == 0 ; sync e_1_down?; assign x_s := 0; },
l_x0_s -> l_x1_s        { guard e_1 == 1-1   and e_2 == 1 ; sync e_1_up?; assign x_s := 0; },
l_x0_s -> l_x0_s        { guard e_2 == 0; sync e_2_down?; },
l_x0_s -> l_x0_s        { guard e_2 == 1; sync e_2_up?; },
l_x0_s -> l_x1_s        { guard e_2 == 1-0   and e_1 == 0 ; sync e_2_down?; assign x_s := 0; },
l_x0_s -> l_x1_s        { guard e_2 == 1-1   and e_1 == 1 ; sync e_2_up?; assign x_s := 0; },

l_x0_s -> l_f_s         { guard x_s >= delta0_l_s; sync s_down!; assign s := 0; },

l_x1_s -> l_x1_s        { guard e_1 == 0; sync e_1_down?; },
l_x1_s -> l_x1_s        { guard e_1 == 1; sync e_1_up?; },
l_x1_s -> l_x0_s        { guard e_1 == 1-0   and e_2 == 1 ; sync e_1_down?; assign x_s := 0; },
l_x1_s -> l_x0_s        { guard e_1 == 1-1   and e_2 == 0 ; sync e_1_up?; assign x_s := 0; },
l_x1_s -> l_x1_s        { guard e_2 == 0; sync e_2_down?; },
l_x1_s -> l_x1_s        { guard e_2 == 1; sync e_2_up?; },
l_x1_s -> l_x0_s        { guard e_2 == 1-0   and e_1 == 1 ; sync e_2_down?; assign x_s := 0; },
l_x1_s -> l_x0_s        { guard e_2 == 1-1   and e_1 == 0 ; sync e_2_up?; assign x_s := 0; },

l_x1_s -> l_f_s         { guard x_s >= delta1_l_s; sync s_up!; assign s := 1; },
```

```
l_f_s -> l_f_s          { guard e_1 == 0; sync e_1_down?; },
l_f_s -> l_f_s          { guard e_1 == 1; sync e_1_up?; },
l_f_s -> l_x1_s         { guard e_1 == 1-0   and e_2 == 0 ; sync e_1_down?; assign x_s := 0; },
l_f_s -> l_x0_s         { guard e_1 == 1-0   and e_2 == 1 ; sync e_1_down?; assign x_s := 0; },
l_f_s -> l_x1_s         { guard e_1 == 1-1   and e_2 == 1 ; sync e_1_up?; assign x_s := 0; },
l_f_s -> l_x0_s         { guard e_1 == 1-1   and e_2 == 0 ; sync e_1_up?; assign x_s := 0; },
l_f_s -> l_f_s          { guard e_2 == 0; sync e_2_down?; },
l_f_s -> l_f_s          { guard e_2 == 1; sync e_2_up?; },
l_f_s -> l_x1_s         { guard e_2 == 1-0   and e_1 == 0 ; sync e_2_down?; assign x_s := 0; },
l_f_s -> l_x0_s         { guard e_2 == 1-0   and e_1 == 1 ; sync e_2_down?; assign x_s := 0; },
l_f_s -> l_x1_s         { guard e_2 == 1-1   and e_1 == 1 ; sync e_2_up?; assign x_s := 0; },
l_f_s -> l_x0_s         { guard e_2 == 1-1   and e_1 == 0 ; sync e_2_up?; assign x_s := 0; }
                                ;
/*-- Instansiation des processus --*
 *--                                              --*/
env_nd_signals_i1 := env_nd_signals ();
Ass_s_i1 := Ass_s ();

/*--    Definition du systeme     --*
 *--                                            --*/
system                                           env_nd_signals_i1,                    Ass_s_i1;
```

B. Les grammaires des descriptions d'entrée de VHDL2TA

B.1. La grammaire des descriptions fonctionnelles en VHDL

vhdl_file : entity architecture ;

entity : ENTITY VHDLID IS PORT '(' ports ')' ';' END [VHDLID] ';' ;

ports : port | ports ';' port ;
port : VHDLID ':' mode BIT ;
mode : IN | OUT ;

architecture : ARCHITECTURE VHDLID OF VHDLID IS def_signals TBEGIN statements END [VHDLID] ';' ;

def_signals : def_signal | def_signals def_signal ;
def_signal : SIGNAL VHDLID ':' BIT ';' ;

statements : ε | statements signal_assignment_statement | statements process_statement ;

signal_assignment_statement : VHDLID AFFECT expr1 ;

expr1 : expr | expr op2 expr ;
expr : BITVALUE | VHDLID | '(' expr op2 expr ')' | NOT expr | '(' expr ')' ;
expr : VHDLID EQ BITVALUE ;

process_statement : process_label ':' PROCESS '(' signals_names_list ')' TBEGIN process_statement_part END PROCESS [process_label] ';' ;

process_statement_part : if_statement ;
if_statement : IF if_statement_ END IF ';' ;
*if_statement_ : condition THEN signal_assignment_statement ELSIF if_statement_*
| condition THEN signal_assignment_statement
| condition THEN signal_assignment_statement ELSE signal_assignment_statement
;

condition : guard_expr1 ;
guard_expr1: guard_expr | guard_expr op2 guard_expr ;
guard_expr : VHDLID EQ BITVALUE
| '(' guard_expr op2 guard_expr ')'
| NOT guard_expr

| '(' guard_expr ')'
;

signals_names_list : signal_name | signals_names_list ',' signal_name ;

process_label : VHDLID ;
signal_name : VHDLID ;

op2 : AND | OR | XOR ;

B.2. La grammaire des descriptions temporelles

Comme on a vu auparavant, il existe deux types de descriptions temporelles que l'outil VHDL2TA prend en entrée :

1. les descriptions d'intervalles des délais associés à chaque porte du circuit à modéliser (donné sous forme d'un fichier).

2. Les descriptions dans lesquelles on associe un fichier de timing (configuration, front-sortie, temps) pour chaque porte du circuit. Ces fichiers sont accessibles via les informations mentionnées dans un fichier principal (représentant de la description temporelle).

Ci-dessous, on présente les deux grammaires associées à ces deux modèles.

B.2.1 La grammaire du premier modèle :

temp_file : delay_signals ;

delay_signals : delay_signal | delay_signals delay_signal ;
delay_signal : signal ':' time_front_dn ',' time_front_up ';'
| signal ':' time_front_up ',' time_front_dn ';'
;

time_front_dn : time dn | '[' time ',' time ']' up ;
time_front_up : time up | '[' time ',' time ']' dn ;

signal : VHDLID ;
time : NUMBER ;
up : 'up' | 'UP' | '+' ;
dn : 'down' | 'DOWN' | 'dn' | 'DN' | '-' ;

B.2.2. La grammaire du deuxième modèle :

Grammaire A (fichier principal) :

temp_file : temp_signals ;

temp_signals : temp_signals temp_signal | temp_signal ;
temp_signal : signal timing_file_signal ;

signal : VHDLID ;
timing_file_signal : FILE_NAME ;

Grammaire B (fichier de timing de portes) :

time_file : sorties ;

sorties : sorties sortie | sortie ;

sortie : param configurations ;

param : signals_in out_in TIME ;

signals_in : signal | signals_in signal ;

configurations : configuration | configurations configuration ;

configuration : n_io time

n_io : n_io io | io ;
io : '0' | '1' | dn | up ;

signal : VHDLID ;
time : NUMBER ;
up : 'u' | 'U' ;
dn : 'd' | 'D' ;

B.3. La grammaire des descriptions d'environnement :

env_file : type_env behaviors_env ;

type_env : [CYCLIC] AUTOMATON ENVIRONMENT
| [AUTOMATON] CYCLIC ENVIRONMENT
| AUTOMATA ENVIRONMENT
| ε
;

behaviors_env : [clock_def] signals_behavior ;

clock_def : CLOCK signal WITH THI time TLO time [INVERSE] [NCYCLES ENTIER] ';' ;

signals_behavior : signal_behavior | signals_behavior signal_behavior ;

signal_behavior : signal ':' time_fronts ';'
| signal ':' value ';'
;

time_fronts : time_front | time_fronts ',' time_front ;

time_front : time front | '[' time ',' time ']' front ;

front: up | dn ;
signal : VHDLID ;
time : NUMBER ;
value : '0' | '1' ;
up : 'up' | 'UP' | '+' ;
dn : 'down' | 'DOWN' | 'dn' | 'DN' | '-' ;

B.4. La grammaire des propriétés TCTL des spécifications en UPPAAL (extraite de [LPY.97])

Prop : E<> StateProp | A[] StateProp ;
StateProp : AtomicProp | (StateProp)
| not StateProp
| StateProp or StateProp
| StateProp and StateProp
| StateProp imply StateProp ;
AtomicProp : Id.Id | Id RelOP Nat | Id RelOP Id Op Nat ;
RelOp : < | <= | >= | > | == ;
Op : + | - ;
Id : Alpha | Id AlphaNum ;
Nat : Num | Num Nat ;
Alpha : A | . . . | Z | a | . . . | z ;
Num : 0 | . . . | 9 ;
AlphaNum : Alpha | Num | - ;